The Evolution of Robust, Reversible, Nano-Scale, Femto-Second-Switching Circuits

Hugo de GARIS

Utah State University
Logan, Utah, USA.
degaris@cs.usu.edu

Evolvable Hardware Conference
June, 2004

The Problem :

How to evolve nano scale circuits that switch in femto-seconds, that don't generate too much heat, that are **robust** (and 3D)?

Moore's Law in 2020 (only ~15 years away) :

If Moore's law continues for another decade or two, then we will be storing a bit of information on a single atom.

This has enormous consequences for electronics, e.g.

a) How to manage the design and fabrication of
"Avogadro Machines?" (with 10^{24} components).

b) How to make Avogadro circuits robust against inevitable fabrication faults?

c) How to make nano circuits compute reversibly (to avoid the heat generation problem)?

d) How to compute when the switching speeds get down to femto-seconds?

e) How to construct such nano circuits? Self assembly? Embryo-genesis? Evolutionary Engineering?

f) How to compute locally, abandoning the “central program memory” paradigm.

g) How to program local 3D circuits? Send intersecting signals from the 2D faces of the “cube”?

Assertions :

a) Nano scale computing will need to be reversible.

Traditional computing (e.g. using NAND gates on chips) is irreversible, hence generates heat (according to Landauer's Principle).

Landauer's Principle = wiping out bits (destroying information) generates heat.

Nano-scale irreversible circuits will have the temperatures of exploding dynamite. Hence we must compute reversibly.

How? By using circuits, using logic gates, that do not destroy bits (information), e.g. Fredkin gate, Toffoli gate.

Reversible means that one can deduce the inputs from the outputs and vice versa (1 to 1), hence the number of inputs and outputs must be the same.

A Toffoli gate has 3 inputs A, B, C, and 3 outputs, X, Y, Z.
If A=1, and B=1, Z= notC, else C. X=A and Y=B.
(Controlled Controlled Not “CCNot” gate).

A Fredkin gate has 3 inputs A, B, C, and 3 outputs, X, Y, Z.
If A=1, Y=C, Z= B, else Y=B, Z=C. X=A.
(Controlled swap gate).

Both gates are reversible and computationally universal.

To build a reversible circuit, use reversible gates. Never destroy bits. Keep a complete history of calculations, get the answer, store it, and then reverse all the output bits. Arrive at start state.

b) Femto-second switching will destroy the old "fetch-execute" paradigm of computing. Computing will have to be local.

Quantum optics is already switching in femto-seconds (a millionth of a nanosecond).

Light travels a foot (30 cms) in a nanosecond, i.e. at today's electronic speeds. In a femto-second, light travels 0.3 microns, i.e. a length of about 300 atoms, i.e. about 30 million atoms in a femto-switching cube of "signal-able" space.

Hence the stored program will have to be distributed throughout the computational medium. It will have to be LOCAL, not in a central program memory that is too far away !!! *Today's computing must die!*

This localized style of computing is called "**Crystalline Computing**".

c) Nano circuits will be impossible to make perfectly, so will have to be fault tolerant. How to build in fault tolerance and at the same time make them reversible?

One possible answer to the above question is the topic of this paper.

d) Nano circuits will need evolutionary engineering techniques for their construction.

With local programming, i.e. storing instructions at each “nano-cube” in the computational medium, predicting the results will be virtually impossible. So how to change the initial programming?

Use an evolutionary engineering approach to nano computing (evolved emergent computing).

d) Programming a 3D space can only be done from a 2D interface.

To store a state at point (x,y,z) send in two signals from 2 faces of a “cube”, so that they intersect at that point simultaneously. The coincidence of the 2 signals at a point sets a bit at that point.

==========

This paper suggests a way to evolve reversibly, and robustly.

Basic idea for reversible evolution. Evolve a ***Boolean network.***

A Boolean node has N bit inputs, and N bit outputs. The node uses a LUT (look up table), that is reversible, i.e. 1 to 1 (bijective). The LUT has 2^N entries in its truth table.

Give each of M nodes in the Boolean network, an initial random 1:1 LUT.

Then evolve the connections between the nodes, AND the output values in the LUTs.

You will need a connection matrix for the network. If the jth output bit of the kth node is connected to the mth input bit of the nth node, then the connection matrix element (kM+j, nM+m) is set to 1, else 0.

A chromosome consists of the LUTs of the nodes and the Connection Matrix (CM).

How to measure the fitness of a Boolean network?

One idea is to choose an arbitrary output bit line and convert its time dependent bit signals to an analog format using a digital convolution window.

The fitness is the degree of match between the output signal and the target signal.

This is a bit crude, but the Boolean network does evolve.

How to make it robust?

By using redundancy. Each previous node now has 3 such “identical” (sub)nodes.

3 copies of each bit are sent to the sub-nodes (1 bit per sub-node).

These 3 bits are input thru a majority selector gate. If >= 2 have the same value, all of the 3 output bits take that value.

These majority selection (MS) gates are placed before and after the node's LUT, to correct single bit errors that occur along the bit lines, AND errors occurring in the LUT itself.

MS gates are sandwiched before and after the nodes, for reversible operation.

This MS gate's truth table is not reversible, so when an error occurs, reversibility is compromised, but this occurs rarely, so the Boolean network is mostly reversible, hence generates little heat.

The network will be error free if there are no "majority" or concurrent errors. At an error rate of about 0.5%, a faulty and error free Boolean network gave **identical** output curves. See paper for details.